On real-time partitioned multicore systems*

Juan Zamorano
jzamora@datsi.fi.upm.es

Juan A. de la Puente
jpuente@dit.upm.es

Universidad Politécnica de Madrid (UPM), Spain

Abstract

Partitioning is a common approach to developing mixed-criticality systems, where partitions are isolated from each other both in the temporal and the spatial domain in order to prevent low-criticality subsystems from compromising other subsystems with high level of criticality in case of misbehaviour. The advent of many-core processors, on the other hand, opens the way to highly parallel systems in which all partitions can be allocated to dedicated processor cores. This trend will simplify processor scheduling, although other issues such as mutual interference in the temporal domain may arise as a consequence of memory and device sharing. The paper describes an architecture for multi-core partitioned systems including critical subsystems built with the Ada Ravenscar profile. Some implementation issues are discussed, and experience on implementing the ORK kernel on the XtratuM partitioning hypervisor is presented.

1 Introduction

Mixed-criticality systems are composed of several subsystems, some of which may have a high level of criticality, possibly requiring certification with respect to some domain-specific standard, while some others are not so critical and can be developed using a less demanding verification and validation (V&V) process. Since certification is generally carried out at system level, unless specific arrangements are made all the system components must be certified to the highest criticality level in the system. This is often unfeasible, as lower criticality subsystems may include COTS components that are not amenable to a strict V&V process, and almost always unpractical, as the cost of certification may be unacceptably high.

Although other approaches are possible (see e.g. [3]), partitioning is a common approach to overcoming this kind of problem. It is based on *temporal and spatial isolation* (TSI). Subsystems with different levels of criticality are built in such a way that the temporal behaviour of a subsystem does not affect that of other subsystems (temporal separation), and its use of memory is limited to its own memory space, without invading other subsystems' spaces (spatial separation). In this way, critical subsystems can be certified independently of non-critical subsystems, as a possible misbehaviour in the latter cannot compromise the properties of critical parts of the system.

This principle can be implemented in different ways. A radical approach is *physical separation*, by implementing critical and non-critical subsystems on different hardware platforms, possibly communicating by means of well defined links. However, as more and more powerful processors have become available, there is a trend towards implementing mixed-criticality systems on a single computer platform. The concept of *Integrated Modular Avionics* (IMA) [1] is a well-known example of this approach that has developed in the aeronautic field. The term *partitioned systems* is commonly applied to this kind of system.

*This work has been partially funded by the Spanish Government, project HI-PARTES (TIN2011-28567-C03-01), and by the European Commission FP7 programme, project MultiPARTES (IST 287702).

Partitioned systems consists of a number of *partitions*, running on a shared computer platform. Each partition hosts a different subsystem with a given criticality level. Partitions are isolated from each other both in the temporal and in the spatial domain, although some means of inter-partition communication are usually provided. A *separation kernel* takes care of implementing temporal and spatial separation by scheduling the execution of partitions and providing separated memory spaces for them. In this way, only the most critical partitions, together with the separation kernel, are required to undergo the certification process.

Virtualization provides a means to divide a single hardware platform into a number of virtual machines, each with a set of virtual resources that are mapped to the available physical resources. This technique can be used to implement partitioning, by making each virtual machine a separate partition, possibly with a different operating system depending on the criticality requirements of the subsystems, or applications, running on it (figure 1). Although there are different approaches to virtualization, it is generally accepted that the best approach for real-time embedded systems is based on the use of a *hypervisor* or virtual machine monitor [16].

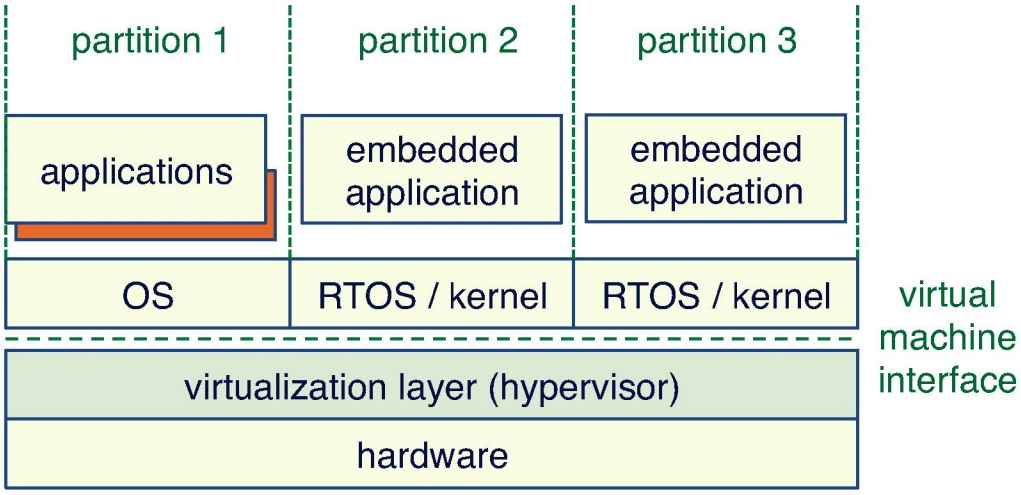

Figure 1: Virtualization and partitions.

In the recent years we have applied these ideas to real-time systems written in Ada with the ORK+ kernel and the XtratuM hypervisor [9, 10, 21]. The resulting ORK+/XtratuM platform is currently being used to develop some pilot real-time systems, with promising results. The platform currently runs on monoprocessor computers, but as multicore processors are becoming more and more common in embedded systems (see e.g. European Comission [11]), the need for extending the concepts of partitioned systems to this kind of system has arisen as a natural extension. In the rest of this paper we analyse the implications of using multicore processors to run partitioned systems, and explore some ideas about the implementation of mixed-criticality real-time systems in Ada on top of such kind of architecture.

2 Multicore processors

In the following we shall assume a multicore processor architecture, where a number of identical processor cores are packaged together in a single chip. All cores share a common memory accessed through a computer bus, and caches may be local or global to all the cores. Cache coherence is ensured by hardware. Other hardware resources, such as timers or bus interfaces, may also be shared.

This kind of architecture is supported by the current Ada standard [2] by means of the packages in the `System.Multiprocessors` hierarchy. A task may be assigned to a dispatching domain, i.e. a set of

processors on which a task may run, and can also be restricted to executing on a particular processor (CPU). In this way, an Ada program can control very precisely the processor on which its tasks run, thus enabling a predictable execution for concurrent and real-time programs [4, 18].

From the real-time systems point of view, multicore architectures pose some important questions that are not fully solved so far:

- Task scheduling methods and resource control protocols have a high complexity and sometimes exhibit unexpected behaviour [8, 12]. Using a fully portioned approach, with every task running always on the same processor, is often used together with FPPS[1] or EDF[2] within each processor in order to enable an analysable temporal behaviour.
- Cache interference and bus contention introduce additional unpredictability and complexity to execution-time analysis [13, 20].
- Shared hardware resources may also add uncertainty to the run-time behaviour of multiprocessor systems.

A version of the Ravenscar profile for multiprocessors using Ada 2012 constructs to provide fully partitioned scheduling has been proposed by Ruiz [17].

3 Partitioned multicore systems

Mixed-criticality systems can be implemented as partitioned systems on top of a multicore platform. Figure 2 shows a partitioned architecture built with a modified version of the XtratuM hypervisor[3] that has been taken as a basis for the HI-PARTES[4] and MultiPARTES[5] projects.

As in the mono processor architecture, the hypervisor provides an interface with a set of virtual machines that are used to hold partitions, possibly with different operating systems and different criticality levels. However, on a multicore platform each partition can run on one or more virtual processors. Several virtual processors can be mapped to a single physical processor, multiplexing its execution time among all the virtual processors, as is the rule in monoprocessor systems. However, provided there are enough cores, each virtual processor can be mapped to a single physical processor core, thus providing physical parallelism to the execution of partitions. Mixed schemes are also possible.

Indeed, as multicore architectures evolve towards a growing number of processor cores, a fully parallel approach is getting most interesting for mixed-criticality systems. Having one or more physical cores dedicated to each partition simplifies scheduling (see below) and eases temporal separation. However, some problems remain that have to be analysed. The main ones are predictable temporal behaviour, virtualization of hardware devices other than processor cores, and interrupt handling. The next paragraphs discuss more in detail these and other topics.

Processor allocation. In order to enable predictable execution, a static assignment of partitions (or virtual processors) to physical cores is assumed.

[1] Fixed-priority pre-emptive scheduling.

[2] Earliest-deadline first.

[3] See `www.xtratum.org` for more details.

[4] www.dit.upm.es/rts/projects/hipartes

[5] `www.multipartes.eu`

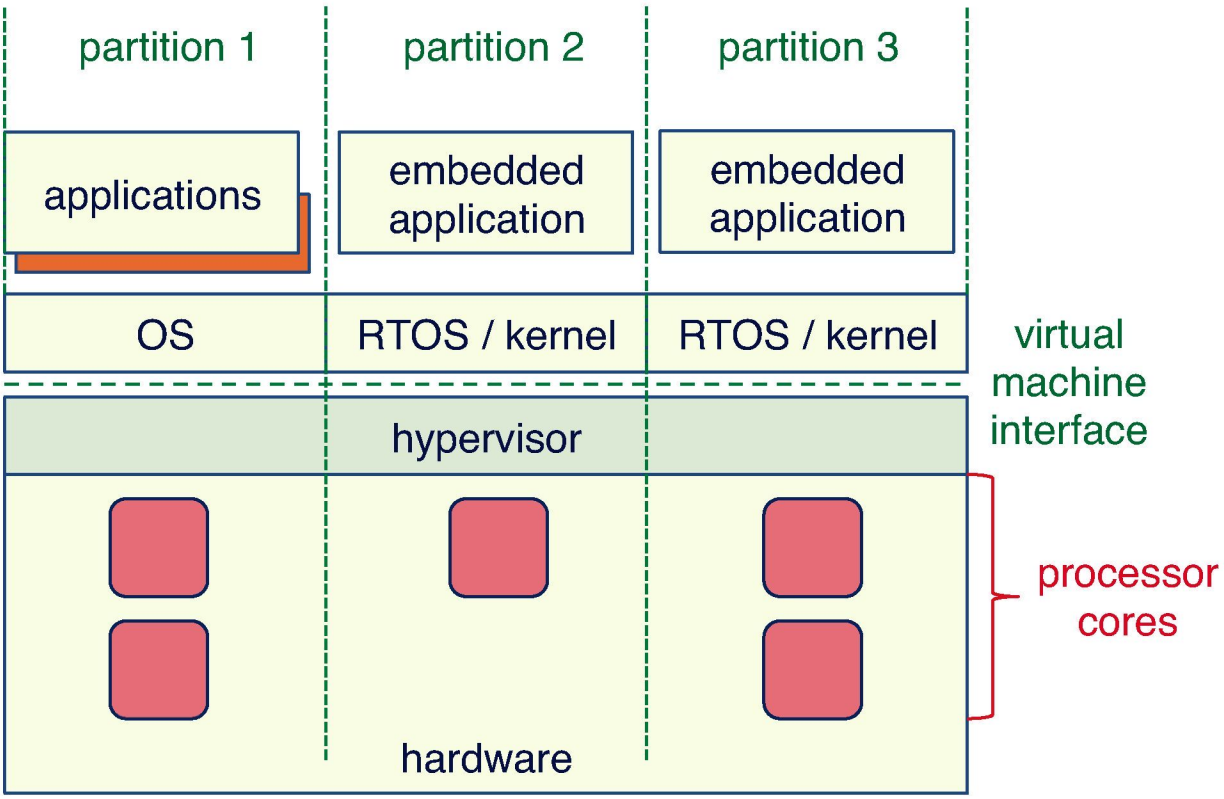

Figure 2: Partitioned multicore systems.

Scheduling. Scheduling in partitioned systems is commonly done in a hierarchical way. A *global scheduler* multiplexes processor time among partitions, whereas a *local scheduler*, usually part of the partition OS or kernel, rules the execution of application processes and tasks within each partition. Although different approaches are possible (see e.g. [14, 15, 19]), we assume here a static, ARINC-like global scheduling method whenever global scheduling is necessary [7].

However, if there are at least as many cores as partitions, as it is likely to become common in the near future, a simpler approach is possible. If each partition is allocated one or more virtual processors that are mapped one-to-one to physical processor cores, there is no need for global scheduling of processor time. All the partitions run in parallel, and only a local scheduler is needed in each partition in order to allocate processor time to the processes or tasks in the partition. Ideally the performance of physically separated systems could be achieved, with a significant reduction in cost. Nevertheless life is not so simple, as partitions still share memory and other hardware resources such as timers and input-output devices.

Temporal separation. Temporal separation can be achieved in a simple way by physical parallelism. If partitions run on separate processors, the temporal behaviour of a partition is independent from other partitions. However, the calculation of WCET must take into account the interference induced by other partitions accessing shared memory or devices. Whenever possible, virtual devices should be assigned dedicated hardware devices in order to reduce such kind of interference.

Spatial separation. Separation of memory spaces between partitions can be enforced by using hardware memory management units (MMU), as in monoprocessor systems.

Interrupt handling. Interrupt sources can be statically assigned to cores by using the programmable interrupt controller (PIC). Notice that input-output registers in the interrupting device can be allocated to the partition to which the core is assigned. In this way, the device can be handled by the partition software without the needed for using hypercalls to manage interrupts or to access registers. Special cases are the programmable interrupt timers, that can be statically assigned to partitions so that partition timing services can be implemented efficiently.

4 Implementing the Ravenscar profile on partitioned multicore systems

The Ravenscar profile was originally designed to achieve predictable tasking in Ada real-time systems [5]. It has found wide acceptation in industry and academy, and has been incorporated into the Ada standard since the 2005 amendment. Since it was developed for monoprocessor platforms, its use in multiprocessor and partitioned systems has to be analysed and, where necessary, implementation mechanisms have to be devised.

Ruiz [17] discussed the extension of the profile to multiprocessor systems. Fully-partitioned fixed-priority pre-emptive scheduling is proposed in order to enable the temporal behaviour to be analysed, in spite of the reduced worst-case schedulable utilization that can arise. The `CPU` aspect can be used to statically allocate a task to a processor. This proposal was implemented in a version of GNAT for LEON computers [6]. Protected objects used by tasks on different processors were implemented using fair locks (i.e. locks with a bounded blocking time), and interrupt handler affinities are set by the startup routine as there is no high-level mechanism to do it in Ada.

Zamorano *et al.* [21] considered the extension of Ada real-time services to partitioned systems, and developed an implementation of the Ravenscar profile on top of the XtratuM hypervisor [9].

A natural extension based on these two approaches can then be proposed in order to implement the Ravenscar profile on partitioned multicore systems. Its basic elements are:

- Partitions are statically allocated to one or more virtual processors, which are mapped one-to-one to physical processors.
- Each partition has an isolated memory space. Isolation is managed by the hypervisor using the underlying MMU hardware.
- Ravenscar partitions are always active and run a single Ada program compiled with the Ravenscar profile. Tasks within the program are statically allocated to processors and are scheduled with a fixed-priority pre-emptive method, as prescribed by the profile.
- Protected operations are accessed using a fair lock, and executed in the processor of the calling task.
- Input-output devices allocated to the partition are not shared with other partitions.
- Interrupt handlers are statically allocated to processors by the hypervisor.
- Dedicated hardware timers are allocated to the partition in order to provide timing services.

An open issue is whether enforcing the Ravenscar profile in a partition is enough to guarantee a fully predictable temporal behaviour. Interference from cache and memory access from other partitions, interrupt handling, and other factors, may influence the execution time of the tasks running in a Ravenscar partition. Another important question is what level of certification can be achieved in a given partition. We are working on a pilot implementation using ORK+ and XtratuM on a multicore LEON3 computer, which we expect will contribute to provide answers to these and other questions.

5 Conclusions and future work

The trend towards using multicore processors in embedded systems, and the need to run applications with mixed criticality levels on the same computer platform lead naturally to the concept of partitioned multicore systems. The mid-term scenario is one where there are more processor cores than partitions, thus making unnecessary the use of partition scheduling methods such as the kind of static scheduling used in ARINC-653 and other architectures.

We have analysed the implications of such architecture for developing real-time systems using Ada and the Ravenscar profile, and have proposed an implementation scheme based on the assumption that at least one processor core can be exclusively allocated to every partition. Although there are other possible approaches to building partitioned many-core systems, we believe that our proposal is simple to implement and ensures a high level of temporal predictability to critical partitions.

We are working on a pilot implementation of such a system, and we expect to derive additional knowledge from it, including metrics and an assessment of the criticality levels that can be successfully guaranteed with our approach.

Acknowledgments

The authors wish to acknowledge the fruitful collaboration with the members of the Hi-PARTES and MultiPARTES teams.

References

[1] ARINC653. *Avionics Application Software Standard Interface — ARINC Specification 653-1*. ARINC, October 2003.

[2] ARM12. *ISO/IEC 8652:2012(E): Information Technology — Programming Languages — Ada*, 2012.

[3] S. Baruah and A. Burns. Implementing mixed criticality systems in Ada. In A. Romanovsky and T. Vardanega, editors, *Reliable Software Technologies - Ada-Europe 2011*, volume 6652 of *Lecture Notes in Computer Science*, pages 174–188. Springer Berlin Heidelberg, 2011.

[4] A. Burns and A. Wellings. Support for multiprocessor platforms. In *15th International Real-Time Ada Workshop (IRTAW 15)*, September 2011.

[5] A. Burns, B. Dobbing, and G. Romanski. The Ravenscar tasking profile for high integrity real-time programs. In L. Asplund, editor, *Reliable Software Technologies — Ada-Europe'98*, volume 1411 of *Lecture Notes in Computer Science*, pages 263–275. Springer Berlin Heidelberg, 1998. ISBN 978-3-540-64536-8. doi: 10.1007/BFb0055011.

[6] F. Chouteau and J. F. Ruiz. Design and implementation of a Ravenscar extension for multiprocessors. In A. Romanovsky and T. Vardanega, editors, *Reliable Software Technologies — Ada-Europe 2011*, number 6652 in LNCS, pages 31–45. Springer-Verlag, 2011.

[7] A. Crespo, I. Ripoll, and M. Masmano. Partitioned embedded architecture based on hypervisor: The XtratuM approach. In *Dependable Computing Conference (EDCC), 2010 European*, pages 67 –72, april 2010.

[8] R. I. Davis and A. Burns. A survey of hard real-time scheduling algorithms and schedulability analysis techniques for multiprocessor systems. *ACM Computing Surveys*, 43(4), 2011.

[9] A. Esquinas, J. Zamorano, J. A. de la Puente, M. Masmano, I. Ripoll, and A. Crespo. ORK+/XtratuM: An open partitioning platform for Ada. In A. Romanovsky and T. Vardanega, editors, *Reliable Software Technologies — Ada-Europe 2011*, number 6652 in LNCS, pages 160–173. Springer-Verlag, 2011.

[10] A. Esquinas, J. Zamorano, J. A. la Puente, M. Masmano, and A. Crespo. Time and space partition platform for safe and secure flight software. In *Data Systems in Aerospace — DASIA 2012*, Dubrovnik, Croatia, 2012.

[11] European Comission. Mixed criticality systems, 2012.

[12] S. Lin, A. Wellings, and A. Burns. Supporting lock-based multiprocessor resource sharing protocols in real-time programming languages. *Concurrency and Computation: Practice and Experience*, 2012.

[13] E. Mezzetti, A. Betts, J. Ruiz, and T. Vardanega. Cache-aware development of high-integrity systems. In J. Real and T. Vardanega, editors, *Reliable Software Technologiey — Ada-Europe 2010*, volume 6106 of *Lecture Notes in Computer Science*, pages 139–152. Springer Berlin / Heidelberg, 2010. ISBN 978-3-642-13549-1.

[14] M. Mollison, J. Erickson, J. Anderson, S. Baruah, and J. Scoredos. Mixed-criticality real-time scheduling for multicore systems. In *Computer and Information Technology (CIT), 2010 IEEE 10th International Conference on*, pages 1864 –1871, July 2010.

[15] J. A. Pulido, S. Urueña, J. Zamorano, T. Vardanega, and J. A. de la Puente. Hierarchical scheduling with Ada 2005. In L. M. Pinho and M. González-Harbour, editors, *Reliable Software Technologies — Ada-Europe 2006*, volume 4006 of *LNCS*. Springer Berlin/Heidelberg, 2006. ISBN 3-540-34663-5.

[16] M. Rosenblum and T. Garfinkel. Virtual machine monitors: current technology and future trends. *Computer*, 38(5):39 – 47, may 2005. ISSN 0018-9162. doi: 10.1109/MC.2005.176.

[17] J. F. Ruiz. Towards a ravenscar extension for multi-processor systems. *Ada Letters*, XXX(1):86—-90, April 2010. 14th International Real-Time Ada Workshop (IRTAW 14).

[18] J. F. Ruiz. Going real-time with Ada 2012 and GNAT. In *15th International Real-Time Ada Workshop (IRTAW 15)*, September 2011.

[19] S. Vestal. Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance. In *Real-Time Systems Symposium, 2007. RTSS 2007. 28th IEEE International*, pages 239 –243, December 2007.

[20] R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, G. Bernat, C. Ferdinand, R. Heckmann, T. Mitra, F. Mueller, I. Puaut, P. Puschner, J. Staschulat, and P. Stenström. The worst-case execution-time problem—overview of methods and survey of tools. *ACM Trans. Embed. Comput. Syst.*, 7(3):1–53, 2008. ISSN 1539-9087. doi: http://doi.acm.org/10.1145/1347375.1347389.

[21] J. Zamorano, Ángel Esquinas, and J. A. de la Puente. Ada real-time services and virtualization. In *15th International Real-Time Ada Workshop (IRTAW 15)*, September 2011.